BookBazaar - Library Management and Review System

Table of Contents

---

1. Introduction

Project Goals

The **BookBazaar - Library Management and Review System** is designed to:

- Manage a library of books using SQLite.

- Allow users to add, update, and delete reviews for books using MongoDB.

- Provide a RESTful API for interacting with the system.

Technologies Used

- **Backend:** Flask (Python)

- **Database:** SQLite (for book management), MongoDB (for reviews)

- **API Testing:** Postman, Web Server

- **Deployment:** Apache

2. Installation and Setup

Python Dependencies

1. Install Python 3.11 or later.

2. Install the required Python packages:

```
pip install Flask pymongo sqlite3
```

SQLite Setup

1. Create a SQLite database named `bookbazaar.db`.

2. Create the (Users, Authors, Books) tables:

```
import sqlite3

# Connect to SQLite (or create it if it doesn't exist)
conn = sqlite3.connect('bookbazaar.db')
cursor = conn.cursor()

# Create Users table
cursor.execute('''
    CREATE TABLE IF NOT EXISTS Users (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        username TEXT NOT NULL UNIQUE,
        email TEXT NOT NULL UNIQUE,
        password TEXT NOT NULL
    )
''')

# Create Authors table
cursor.execute('''
    CREATE TABLE IF NOT EXISTS Authors (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        name TEXT NOT NULL,
        country TEXT NOT NULL
    )
''')

# Create Books table
cursor.execute('''
    CREATE TABLE IF NOT EXISTS Books (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        title TEXT NOT NULL,
        author_id INTEGER NOT NULL,
        genre TEXT NOT NULL,
        published_year INTEGER NOT NULL,
        FOREIGN KEY (author_id) REFERENCES Authors(id)
    )
''')

# Commit changes and close the connection
conn.commit()
conn.close()
print("Database setup complete!")
```

3. Insert sample data:

```
conn = sqlite3.connect('bookbazaar.db')
cursor = conn.cursor()

# Insert sample authors
cursor.execute("INSERT INTO Authors (name, country) VALUES
('J.K. Rowling', 'UK')")
cursor.execute("INSERT INTO Authors (name, country) VALUES
('J.R.R. Tolkien', 'UK')")

# Insert sample books
cursor.execute("INSERT INTO Books (title, author_id, genre,
published_year) VALUES ('Harry Potter', 1, 'Fantasy',
1997)")
cursor.execute("INSERT INTO Books (title, author_id, genre,
published_year) VALUES ('The Hobbit', 2, 'Fantasy', 1937)")

conn.commit()
conn.close()
print("Sample data inserted!")
```

MongoDB Setup

1. Install MongoDB and start the MongoDB server.

```
# Install PyMongo
!pip install pymongo
```

2. Create a database named `bookbazaar_reviews`.

```python
# MongoDB connection details (admin credentials)
MONGO_URI_ADMIN = "mongodb://localhost:27017/"

# Function to create database and user
def setup_mongodb():
    try:
        # Connect to MongoDB as admin
        client = MongoClient(MONGO_URI_ADMIN)
        db = client.bookbazaar_reviews  # Create the database

        # Create a user for the database
        db.command("createUser", "bookbazaar_user", pwd="userpassword", roles=["readWrite"])
        print("Database 'bookbazaar_reviews' and user 'bookbazaar_user' created successfully!")
    except OperationFailure as e:
        print(f"Failed to create database or user: {e}")
    except ConnectionFailure as e:
        print(f"Failed to connect to MongoDB: {e}")

# Run the setup
setup_mongodb()
```

3. Create a collection named `reviews`.

```python
# MongoDB connection details (user credentials)
MONGO_URI = "mongodb://localhost:27017/bookbazaar_reviews"

# Function to connect to MongoDB
def connect_to_mongodb():
    try:
        # Create a connection to MongoDB
        client = MongoClient(MONGO_URI)
        # Ping the server to confirm the connection
        client.admin.command('ping')
        print("Successfully connected to MongoDB!")
        return client
    except ConnectionFailure as e:
        print(f"Failed to connect to MongoDB: {e}")
        return None

# Connect to MongoDB
client = connect_to_mongodb()
```

Apache Configuration (Optional)

Deploying the application using Apache:

1. Install Apache and mod_wsgi.

    - Link: https://httpd.apache.org/docs/2.4/platform/windows.html

2. Configure the Apache virtual host to serve the Flask application.

3. Restart Apache to apply the changes.

3. Running the Application

Starting the Flask Application

1. Navigate to the project directory.

2. Run the Flask application:

```
python app2.py
```

3. The application will start at `http://127.0.0.1:5000`.

```
PS C:\Users\DELL> & C:/Users/DELL/AppData/Local/Microsoft/WindowsApps/python3.11.exe "d:/Sprints/From Data To
 AI/Capstone Project/BookBazaar - Library Management and Review System/app2.py"
Successfully connected to MongoDB!
 * Serving Flask app 'app2'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server
 instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
Successfully connected to MongoDB!
 * Debugger is active!
 * Debugger PIN: 453-298-528
```

Accessing the Application

- Use a web browser or API testing tool (e.g., Postman) to interact with the API.

(Look at Report File)

4. API Documentation

Overview of API Endpoints

HTTP Method	Endpoint	Description
GET	`/books/<id>/reviews`	Get all reviews for a book
POST	`/books/<id>/reviews`	Add a new review for a book
PUT	`/reviews/<review_id>`	Update an existing review
DELETE	`/reviews/<review_id>`	Delete an existing review

Hint: Add (http://127.0.0.1:5000/ Or http:// localhost:5000/) Before Endpoint

Detailed Endpoint Documentation

GET /books/<id>/reviews

- **Description**: Retrieve all reviews for a specific book.

- **Request**:

  - **URL:** `http://127.0.0.1:5000/books/1/reviews`

  - **Method:** `GET`

- **Response**: json

```
[
    {
        "book_id": 1,
        "comment": "Bad book!",
        "rating": 4,
        "user_id": 1
    }
]
```

Sprints / BookBazaar APIs / reviews / **GET Request**

GET http://127.0.0.1:5000/books/1/reviews

Params Auth Headers (6) Body Scripts Settings

Headers 6 hidden

	Key	Value
	Key	Value

Body 200 OK • 55 ms • 257 B

{} JSON Preview Visualize


```
[
    {
        "book_id": 1,
        "comment": "Bad book!",
        "rating": 4,
        "user_id": 1
    }
]
```

POST /books/<id>/reviews

- **Description**: Add a new review for a specific book.

- **Request**:

  - URL: `http://127.0.0.1:5000/books/1/reviews`

  - Method: `POST`

  - **Body (JSON):**

```
{
    "user_id": 1,
    "rating": 5,
    "comment": "Great book!"
}
```

- **Response**: json

```
{
    "message": "Review added",
    "review_id": "65a1b2c3d4e5f6a7b8c9d0e1"
}

// or

{
    "error": "Book not found"
}
```

Sprints / BookBazaar APIs / reviews / **POST Request**

POST http://127.0.0.1:5000/books/3/reviews

Params Auth Headers (9) Body Scripts Settings

raw JSON

```
1 {
2     "user_id": 1,
3     "rating": 5,
4     "comment": "Great book!"
5 }
```

Body 404 NOT FOUND • 6 ms • 204 B

{} JSON Preview Visualize

```
1 {
2     "error": "Book not found"
3 }
```

PUT /reviews/<review_id>

- **Description**: Update an existing review.

- **Request**:

  - **URL**: `http://127.0.0.1:5000/reviews/6785095d1e3ffbdb680e6967`

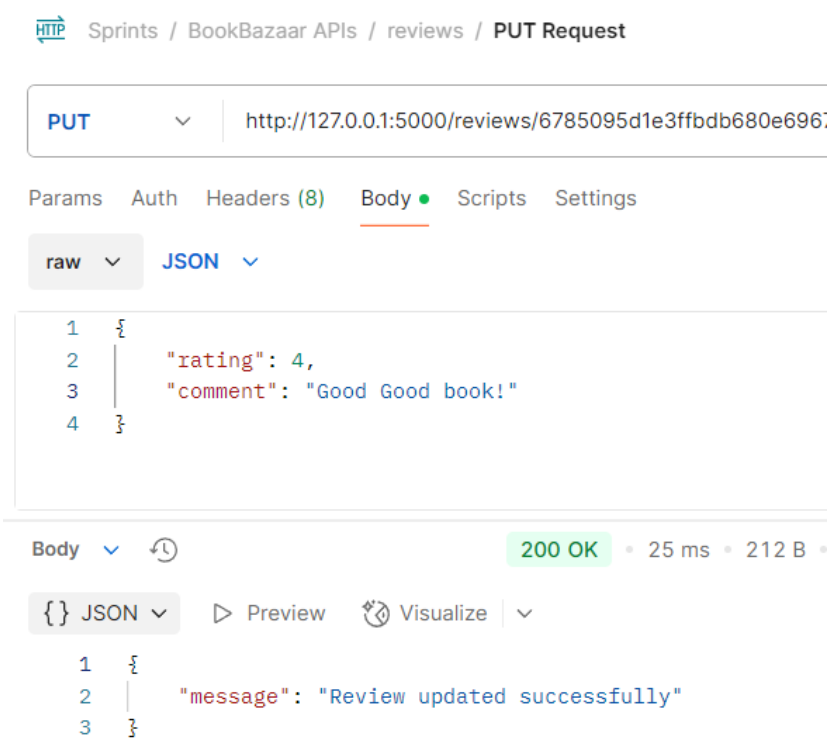
  - **Method**: `PUT`

  - **Body (JSON)**: json

```
{
    "rating": 4,
    "comment": "Good Good book!"
}
```

- **Response**: json

```
{
    "message": "Review updated successfully"
}
```



DELETE /reviews/<review_id>

- **Description**: Delete an existing review.

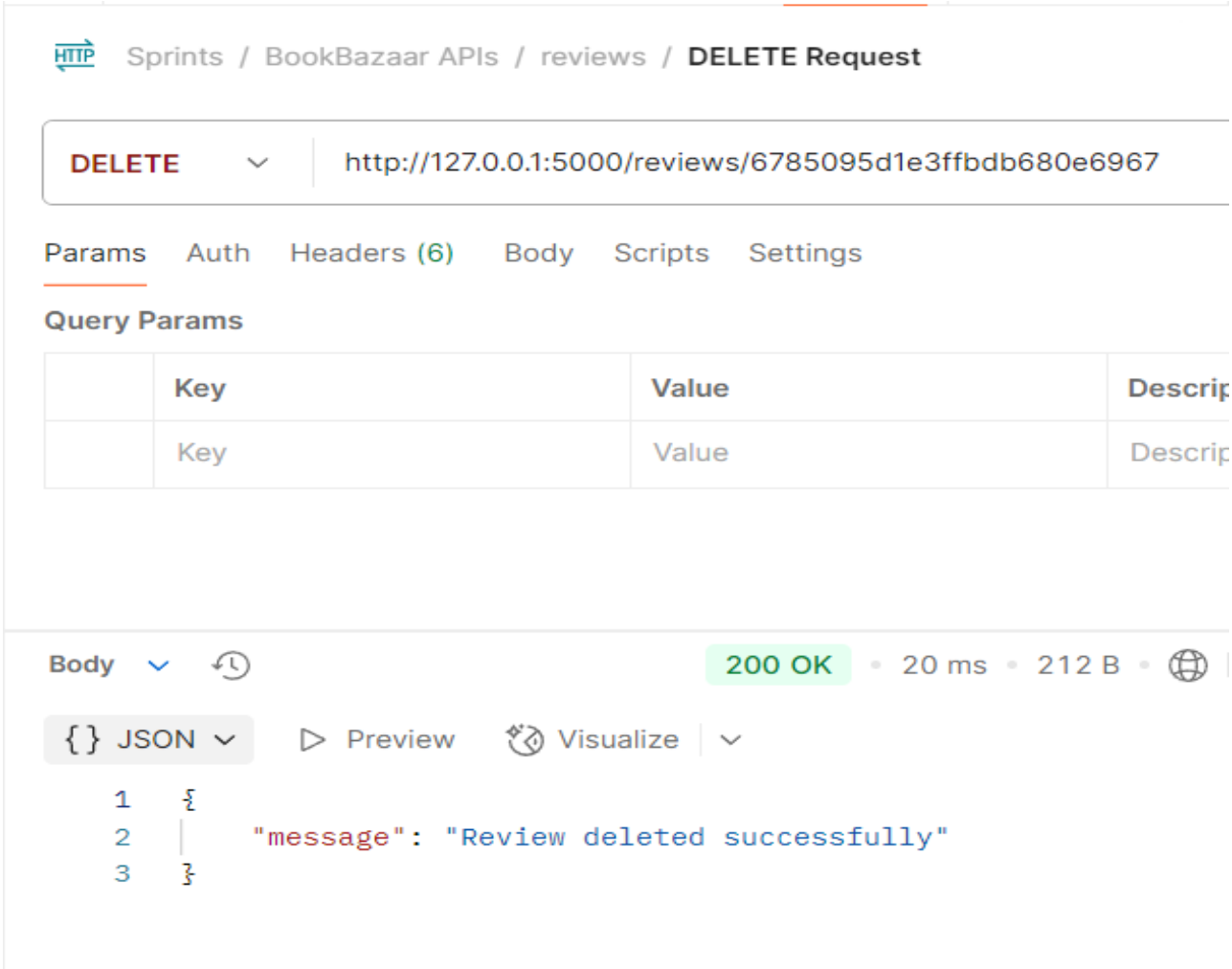
- **Request**:

  - **URL**: `http://127.0.0.1:5000/reviews/6785095d1e3ffbdb680e6967`

  - **Method**: `DELETE`

- **Response**: json

```
{
    "message": "Review deleted successfully"
}
```



5. Postman Testing

Importing Postman Collection

1. Download the Postman collection JSON file for this project.

2. Open Postman and import the collection.

Testing API Endpoints

1. Use the imported collection to test all API endpoints.

2. Modify the request parameters (e.g., `book_id`, `review_id`) as needed.

6. Troubleshooting

Common Issues and Fixes

1. SQLite Error: No such table:

- Ensure the `Books` table exists in the `bookbazaar.db` database.
- Run the `CREATE TABLE` script if the table is missing.

2. MongoDB Connection Error:

- Ensure the MongoDB server is running.
- Verify the connection URI in the Flask application.

3. Flask Application Not Starting:

- Check for syntax errors in the Python code.
- Ensure all dependencies are installed.

7. Conclusion

The **BookBazaar - Library Management and Review System** provides a robust API for managing books and reviews. With clear documentation and examples, new developers can easily set up and use the system.

Attachments

- **Postman Collection**: `Sprints.postman_collection.json`

- **SQLite Database**: `bookbazaar.db`

- **Flask Application**: `app.py`, `app2.py`